Amendments to the Specification:

Please replace the paragraph beginning at page 21, line 18 with the following amended paragraph:

Similarly, a suffix operation [[can be]] is defined as a generic form of computation that takes in $n$ inputs $y_0, y_1, \ldots, y_{n-1}$ and produces $n$ outputs $z_0, z_1, \ldots, z_{n-1}$ according to

Please add the following new paragraphs after the paragraph ending at page 22, line 4:

When a given computational problem can be identified as a prefix or suffix problem well-known structures for executing prefix or suffix operations may be applied. The structures vary in efficiency in either space (e.g., circuit area) or in time (e.g., processing latency). We first address the prefix operation defined in equations (9)-(10). Two well-known structures for executing a prefix operation are the serial prefix and parallel (tree-structured) prefix structures.

A serial prefix computational structure serially computes $y_{[i]}$ from $y_{[i-1]}$. This can have good spatial efficiency since it generally requires only one processor to implement the $\otimes$ operator. However, it can have high latency since $y_{[i]}$ cannot be computed until $y_{[i-1]}$ is available. Specifically, the computational complexity of the serial prefix computational structure is O(1) and the latency is O(n).

A parallel (tree-structured) prefix structure tends to have a larger spatial complexity and lower latency than the serial prefix structure.

Specifically, the spatial complexity is O(n*log(n)) and the latency is O(log(n)). Both serial and parallel prefix structures are well known in the literature. A computational

task most commonly described as a prefix computation is when ⊗ is addition and therefore sums are sought. An associated serial prefix adder circuit is commonly known as a ripple-carry adder (J. M. Rabaey, "Digital Integrated Circuits A Design Perspective" (Prentice Hall Electronics and VLSI Series 1996). An associated parallel (tree-structured) prefix adder circuit is commonly known as a tree-adder.

There are many variations of the parallel prefix structure that are documented in peer reviewed literature (see e.g., R. Zimmermann, Binary Adder Architectures for Cell-Based VLSI and their Synthesis, PhD thesis, Swiss Federal Institute of Technology (ETH) Zurich, Hartung-Gorre Verlag, 1998; B. Parhami, "Computer Arithmetic, Algorithms and Hardware Designs" (Oxford University Press 2000)) and U.S. Patents (see e.g., U.S. Pat. No. 6,628,211). These descriptions are typically based on pictures and examples of parallel prefix structures, although it is well accepted that a general taxonomy of parallel prefix structures exists (see generally Gurkayna, F.K.; Leblebicit, Y.; Chaouati, L.; & McGuinness, P.J.;" Higher Radix Kogge-Stone Parallel Prefix Adder Architectures," 5 IEEE International Symposium on Circuits and Systems, 609 (May 2000)).

Parallel prefix structures share in common the following traits:

1) They accept the vector of inputs $y_{[0]}$, $y_{[1]}$, ... $y_{[n-1]}$ as in (9)- (10).

2) They comprise a number of processing stages.

3) The input to the first processing stage are the inputs $y_{[0]}$, $y_{[1]}$, ... $y_{[n-1]}$ as in (9)-(10).

4) The inputs to stage number k are the outputs of stage number k-1.

5) The processing units within a given stage can be operated in parallel; that is there is no data contingency within a processing stage.

6) The number of processing stages is O(log(n)).

7) The outputs of the final processing stage are $z_{[0]}$, $z_{[1]}$, ... $z_{[n-1]}$.

8) The input data to a processing stage is processed out of natural order. That is, the processing is parallel across the index of the input data.

Since the suffix computational task in (11)-(12) is simply the time reversed version of the prefix computational task in (9)-(10), a given structure for a prefix computation can be used for a suffix computation by reversing the order of the inputs. Therefore, a parallel suffix structure can also satisfy the above properties.

Most discussion in peer-reviewed literature and patents discuss only prefix operations for this reason - e.g., given an structure for a prefix computation, it may be used for a suffix computation..

In the present disclosure, significant computational problems associated with the SISO processing are identified as prefix and suffix operations. The prior art, the forward backward algorithm for SISO processing, can then be interpreted as a *serial* prefix/suffix computation structure. For example, (6) and (7) directly implement the *serial* prefix and suffix structures, respectively.

The present disclosure includes a description of an application of *parallel* (tree-structured) prefix structures based on this identification. A parallel prefix operation is a prefix operation computed by a parallel prefix structure, as described above. Similarly, a parallel suffix operation is a

suffix operation computed by a parallel suffix structure.